

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `ball@berkeley.edu`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

- You must choose either this option
- Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

- You could select this choice.
- You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

0.0.1 CS88 Final Exam.

You can complete and submit these questions before the exam starts.

0.0.2 Clarifications Google Doc

Please open the Google Doc and check for clarifications!

(a)

As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others.

— *UC Berkeley Honor Code*

By typing my name below, I am affirming that all work in this quiz is my own. I have not (and will not) use any resources outside of the allowed course materials, nor will I collaborate with anyone else during the exam.

(b) What is your student ID number?

1. (9.0 points) WWPD

(a) (3.0 pt) Create a linked list called `foo` such that the following statement returns `True`

```
>>> foo = _____(Your answer)_____
>>> len(foo.first) == len(foo.rest.first) and foo.first[0] == "t"
    and foo.rest.rest.rest == Link.empty
True
```

(b) (3.0 points)

Given the lines of code and results, fill in the blanks so that the results appear properly. There are two blanks in this problem.

```
class Candy:
    def __init__(self, color, price):
        self.color = color
        self.price = price

def shop(candies, colors):
    total = 0
    for candy in candies:
        if candy.color in colors:
            print("I like " + candy.color)
            total += candy.price
        else:
            print("I don't like " + candy.color)
    return total

>>> total = shop(_____(1)_____, _____(2)_____)
I don't like blue
I like red
I like green
```

```
I like red  
>>> print(total)  
42
```

i. Blank (1)

ii. Blank (2)

- (c) (3.0 pt) Explain in one sentence (and less than about 25 words) what the following `mystery` function does if given an instance `t` of the `Tree` class. The `Tree` class is also included for reference.

```
class Tree:
    def __init__(self, entry, branches=[]):
        self.entry = entry
        for b in branches:
            assert isinstance(b, Tree)
        self.branches = branches

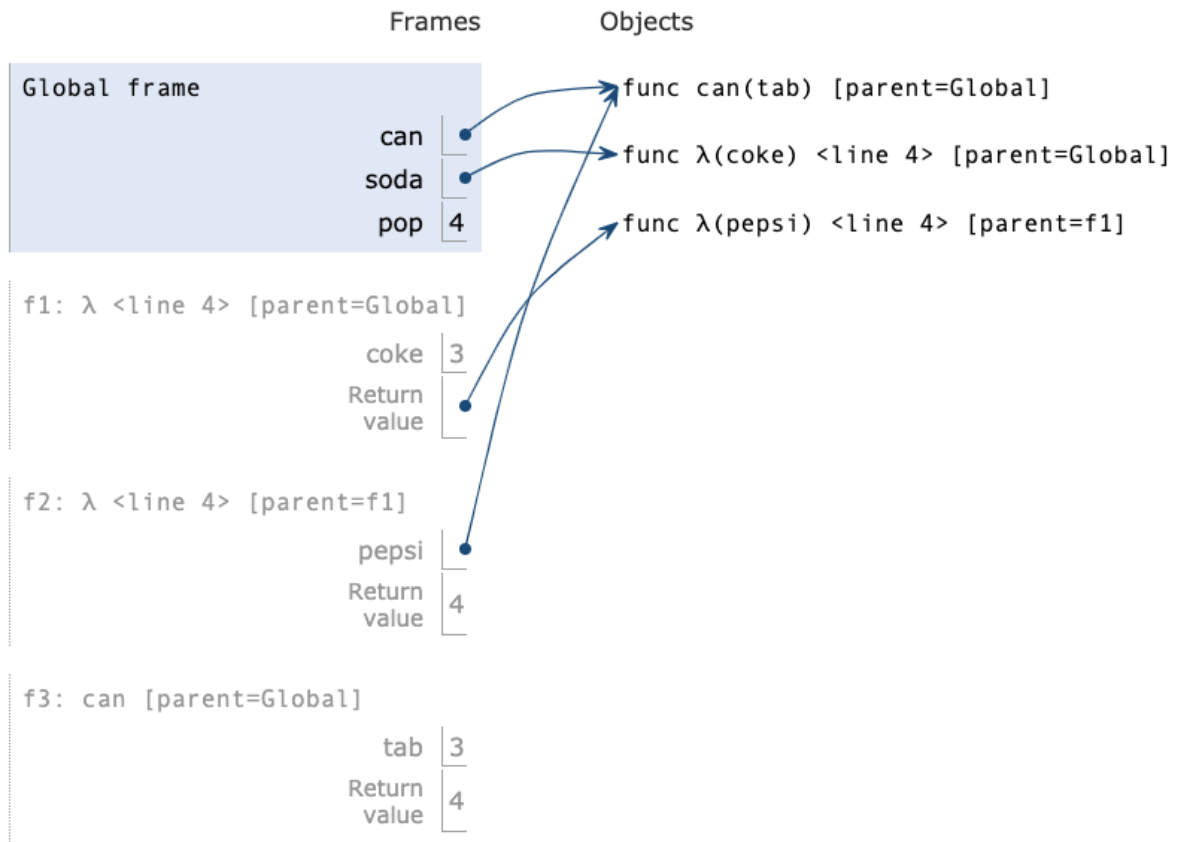
    def is_leaf(self):
        return not self.branches

def mystery(t):
    def hoopla(i, z):
        m = 0
        if i % 2 == 0:
            m += z.entry
        return m + sum([hoopla(i+1, b) for b in z.branches])
    return hoopla(0, t)
```

2. (11.0 points) Reverse Environment Diagram

You are given the environment diagram for each question. Please fill in the blank lines so that the last step of the environment diagram will look exactly the same as the diagram given. All lines must be filled in and cannot be left blank. There is potentially more than one way to do a given problem, but all blanks must be used.

(a) (4.0 pt)

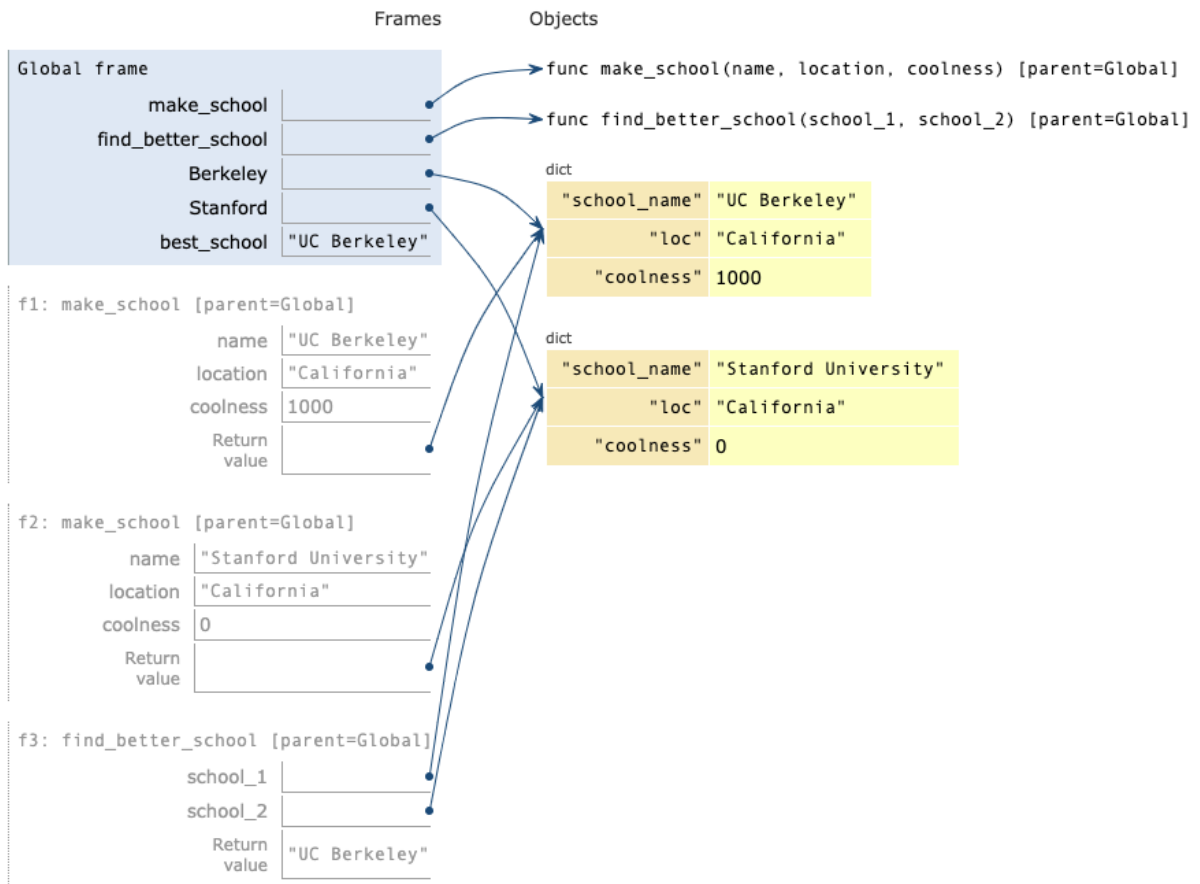


environment diagram for 2.1

```
def can(tab):
    return _____

soda = lambda coke: lambda pepsi: _____
pop = soda(____)(_____)
```



environment diagram for 2.2

(b) (7.0 pt)

```
def make_school(name, location, coolness):
    return _____

def find_better_school(school_1, school_2):
    if school_1["coolness"] > school_2["coolness"]:
        return _____
    else:
        return _____

berkeley = make_school("UC Berkeley", _____, _____)
stanford = make_school("Stanford University", _____, _____)

best_school = find_better_school(berkeley, stanford)
```

HINT: *The dict found in the environment diagram refers to a dictionary instance.*



3. (8.0 points) The Disney LINKup

The `Character` class is given to you below and will be used for this problem.

```
class Character:
    def __init__(self, name, charm):
        self.name = name
        self.charm = charm

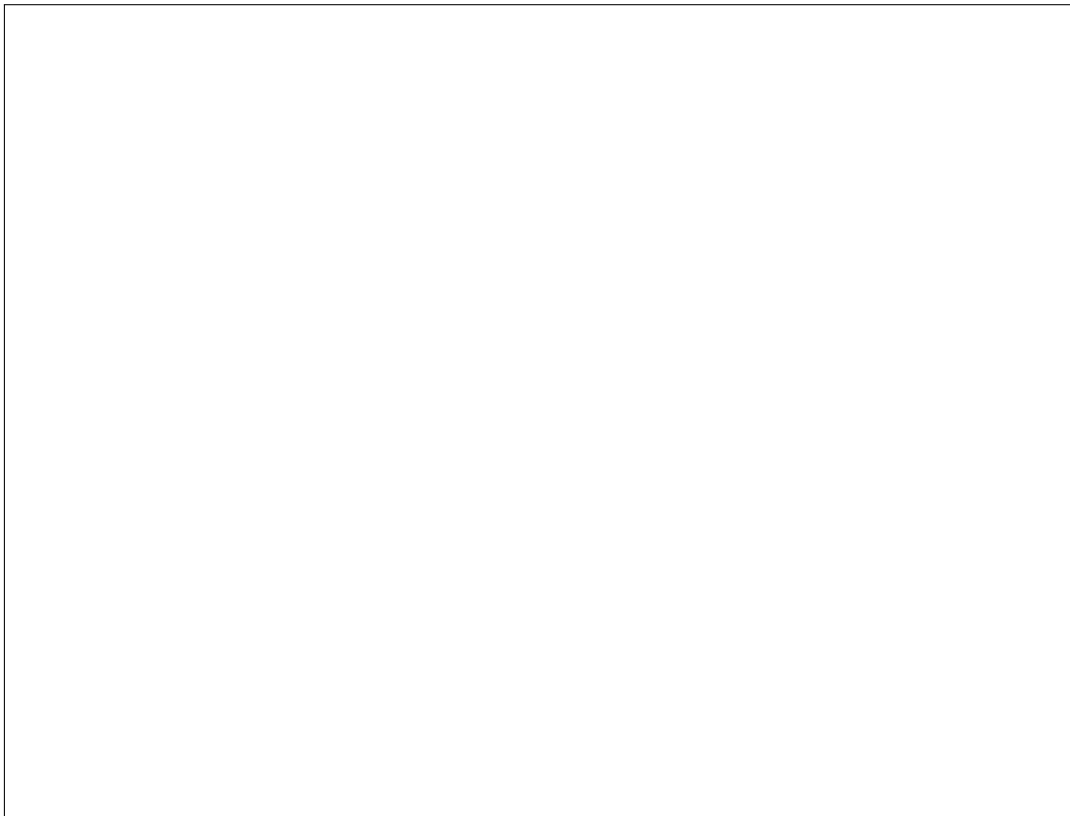
    def __repr__(self):
        return 'Character({0}, {1})'.format(self.name, self.age)
```

- (a) Your favorite Disney characters are preparing for an epic face-off, but you need to pair up characters into teams before the games begin! For this problem, you can imagine that the characters are standing in a line (represented as a linked list).

Given a *linked list* (a `Link`) of characters, return a Python list of pairs, where each pair is a two element list containing both characters on that team. Pairs should be created from left to right. However, if a `Character` is part of a team where their teammate has a greater charm factor than them, this `Character`'s charm factor increases to meet their teammate's!

Before you return the final teams, you should also print whether or not all the teams are full, i.e. if every team has two members. See the doctests on the message to print in either case!

```
def make_pairs(lnk):
    """
    >>> belle = Character("Belle", 35)
    >>> simba = Character("Simba", 45)
    >>> flynn = Character("Flynn", 30)
    >>> teams1 = make_pairs(Link(belle, Link(simba, Link(flynn))))
    Not all teams are full!
    >>> teams1
    [[Character(Belle, 45), Character(Simba, 45)], [Character(Flynn, 30)]]
    >>> mulan = Character("Mulan", 40)
    >>> aladdin = Character("Aladdin", 30)
    >>> olaf = Character("Olaf", 50)
    >>> jafar = Character("Jafar", 5)
    >>> teams2 = make_pairs(Link(mulan, Link(aladdin, Link(olaf, Link(jafar))))))
    All teams full!
    >>> teams2
    [[Character(Mulan, 40), Character(Aladdin, 40)], [Character(Olaf, 50), Character(Jafar, 5)]]
    """
```



4. (10.0 points) **HO**Finetly a safe passcode

- (a) (8.0 pt) Fill out the blanks below to encode a `passcode` by adding numbers to each digit. The input of the `encoder` function is a list of single digit numbers to use to encode a certain `passcode`. The `encoder` function returns a function that will take in the `passcode` to encode.

The way a `passcode` will be encoded is by incrementing each digit in the `passcode` by a number from the variable `lst` passed into the `encoder` function. The function uses a number from `lst` one at a time (in order) to increment each digit in `passcode`, cycling around `lst` if necessary. See the doctests for some examples

```
def encoder(lst):
    """
    Returns a function that will use the input list to encode an input passcode
    >>> increment_password = encoder([3])
    >>> increment_password("111")
    "444"
    >>> encoder([1,2,3])("234")
    "357"
    >>> encoder([5,2])("1111")
    "6363"
    """
    def encode_helper(passcode):
        output = ""
        for i in _____:
            curr = _____(_____) + lst[_____]
            output = _____
        return _____
    return _____
```

- (b) **Runtime!**

For this problem, you may assume `encoder` is implemented correctly.

- i. What is the runtime of the `encoder` function where

n

is the length of `encoder_helper`?

- $\theta(1)$
- $\theta(n)$
- $\theta(n^2)$
- $\theta(2^n)$

- ii. What is the runtime of the `increment` function where

n

is the length of `passcode`?

- $\theta(1)$
- $\theta(n)$
- $\theta(n^2)$
- $\theta(2^n)$

5. (8.0 points) The Silent TREETment

- (a) In this problem, every node in a given Tree has a string as its entry. Your job is to write a function that, given a Tree T, will return its “message” which is defined as the messages of all of T’s branches (in order) followed by T.entry. If T is a leaf node, then T’s message is T.entry.

However, some trees are not being cooperative and giving you the silent treatment - how rude! Any subtree that has an empty string as a message is giving you the silent treatment. If any of the subtrees in a tree T give you the silent treatment, then T’s entire message is an empty string.

Assume that you will be passed a valid tree.

The function `getTreeMsg` should behave as shown below.

```
>>> t = Tree("4", [Tree("2", [Tree("1")]), Tree("3")])
>>> getTreeMsg(t)
"1234"
```

```
>>> t = Tree("4", [Tree("2", [Tree("")]), Tree("3")])
>>> getTreeMsg(t)
""
```

The skeleton code is given below.

```
def getTreeMsg(tree):
    if _____:
        return _____
    msg = _____
    for _____:
        _____
        if _____:
            _____
        msg += _____
    return _____
```

6. (8.0 points) SQL for Sequels

Use the following tables to write queries below. We have given you structure of each query, as a start. **Not all blanks need to be filled in for each query.**

When writing your answer, please only include SQL keywords that you use in your query and leave out any you don't use.

```
CREATE TABLE novels AS
```

```
SELECT "Sorcerer's Stone" as title, 1 as number, "Harry Potter" as series UNION
SELECT "Chamber of Secrets", 2, "Harry Potter" UNION
SELECT "Catching Fire", 2, "Hunger Games" UNION
SELECT "Mockingjay", 3, "Hunger Games" UNION
SELECT "Deathly Hallows", 7, "Harry Potter" UNION
SELECT "Cat in the Hat", 1, "N/A" UNION
SELECT "Tale of Two Cities", 1, "N/A" UNION
SELECT "Divergent", 1, "Divergent" UNION
SELECT "Insurgent", 2, "Divergent" UNION
SELECT "Mark of Athena", 3, "Heroes of Olympus" UNION
SELECT "House of Hades", 4, "Heroes of Olympus" UNION
SELECT "Son of Neptune", 2, "Heoes of Olympus";
```

```
CREATE TABLE series_authors AS
```

```
SELECT "Harry Potter" as series, "JK Rowling" as author, "UK" as country, "Yate" as city UNION
SELECT "Hunger Games", "Suzanne Collins", "USA", "Hartford" UNION
SELECT "Divergent", "Veronica Roth", "USA", "New York" UNION
SELECT "Heroes of Olympus", "Rick Riordan", "USA", "San Antonio";
```

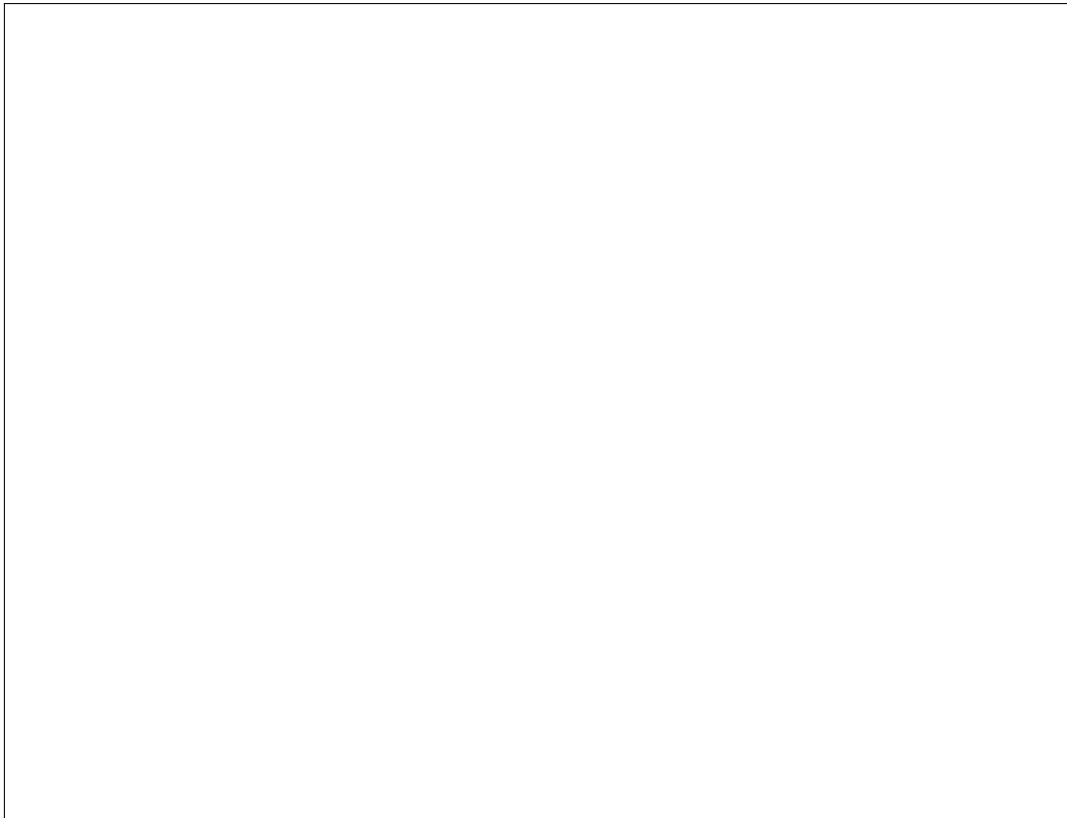
- (a) (2.0 pt) Write a SELECT statement which would output the names of series with *more than* with more than two books in the novels table:

```
SELECT _____  
FROM _____  
WHERE _____  
GROUP BY _____  
HAVING _____  
ORDER BY _____
```

Output:

Harry Potter

Heroes of Olympus



- (b) (3.0 pt) Given the tables above, write a SELECT statement which has each book name with the name of its direct sequel, meaning the book that comes right after it (for example: Book 5 is not Book 2's direct sequel but Book 3 is Book 2's direct sequel). The prequel should be listed before the sequel.

```
SELECT -----  
FROM -----  
WHERE -----  
GROUP BY -----  
HAVING -----  
ORDER BY -----
```

Output:

```
Sorcerer's Stone | Chamber of Secrets  
Catching Fire   | Mockingjay  
Divergent        | Insurgent  
Mark of Athena  | House of Hades  
Son of Neptune  | Mark of Athena
```

- (c) (3.0 pt) Write a `SELECT` statement which would output the book name, author, and author's city of birth for books in the books table that are *the second book* in a series of books *and have authors from the USA*. Order the output by the author's name alphabetically.

```
SELECT -----  
FROM -----  
WHERE -----  
GROUP BY -----  
HAVING -----  
ORDER BY -----
```

Output:

Son of Neptune	Rick Riordan	San Antonio
Catching Fire	Suzanne Collins	Hartford
Insurgent	Veronica Roth	New York

7. (4.0 points) AlTeRnAtInG Generator

- (a) Create a generator that when given 3 generator functions, cycles through yielding from each generator.

```
def alternating_generator(gen1, gen2, gen3):
    """
    >>> def evens():
        "a generator that generates even numbers starting at 2"
    >>> def odds():
        "a generator that generates odd numbers starting at 1"
    >>> def nines():
        "a generator that generates multiples of 9 starting at 9"

    >>> gen = alternating_generator(evens, odds, nines)
    >>> next(gen) # return a value from evens
    2
    >>> next(gen) # return a value from odds
    1
    >>> next(gen) # return a value from nines
    9
    >>> next(gen) # return a value from evens
    4
    >>> next(gen) # return a value from odds
    3
    """
```

Use the following lines to answer complete `alternating_generator`. They are out of order. Note: Lines can be indented as fit, and you might not need all lines. However, you should not use extra lines.

```
while _____:
    for _____ in _____:
        iter_lst = [_____, _____, _____]
        _____next(_____)_____
        _____
        _____
```




8. (12.0 points) Stop ORDERing Me Around

- (a) You work for a shipping company and are making a program that creates an order. Each order contains Packages. Depending on the weight of the item, it either goes into a `LargePackage` or a `SmallPackage`. The cost for shipping for `LargePackages` is 5 and the cost of `SmallPackages` is 2. Each order has a `calculateShippingCost` method that calculates the total shipping cost of the order. It also has an `allItems` method which prints out the names of all the items in the order.

```
>>> newOrder = Order()
>>> newOrder.addItem("Keyboard", 2)
>>> newOrder.addItem("Mouse", 1)
>>> newOrder.addItem("Monitor", 6)

>>> newOrder.calculateShippingCost() # 2 (Keyboard) + 2 (Mouse) + 5 (Monitor) = 9
9

>>> newOrder.allItems()
Keyboard
Mouse
Monitor

>>> SmallPackage.shippingCost = 3
>>> newOrder.calculateShippingCost() # 3 (Keyboard) + 3 (Mouse) + 5 (Monitor) = 11
11
```

Given the above behavior, complete the `Order`, `Package`, `LargePackage`, and `SmallPackage` classes

```
class Order:

    def __init__(self):
        -----
        -----

    def addItem(self, item, weight):
        -----
        -----
        -----

    def calculateShippingCost(self):
        -----
        -----
        -----

    def allItems(self):
        -----
        -----

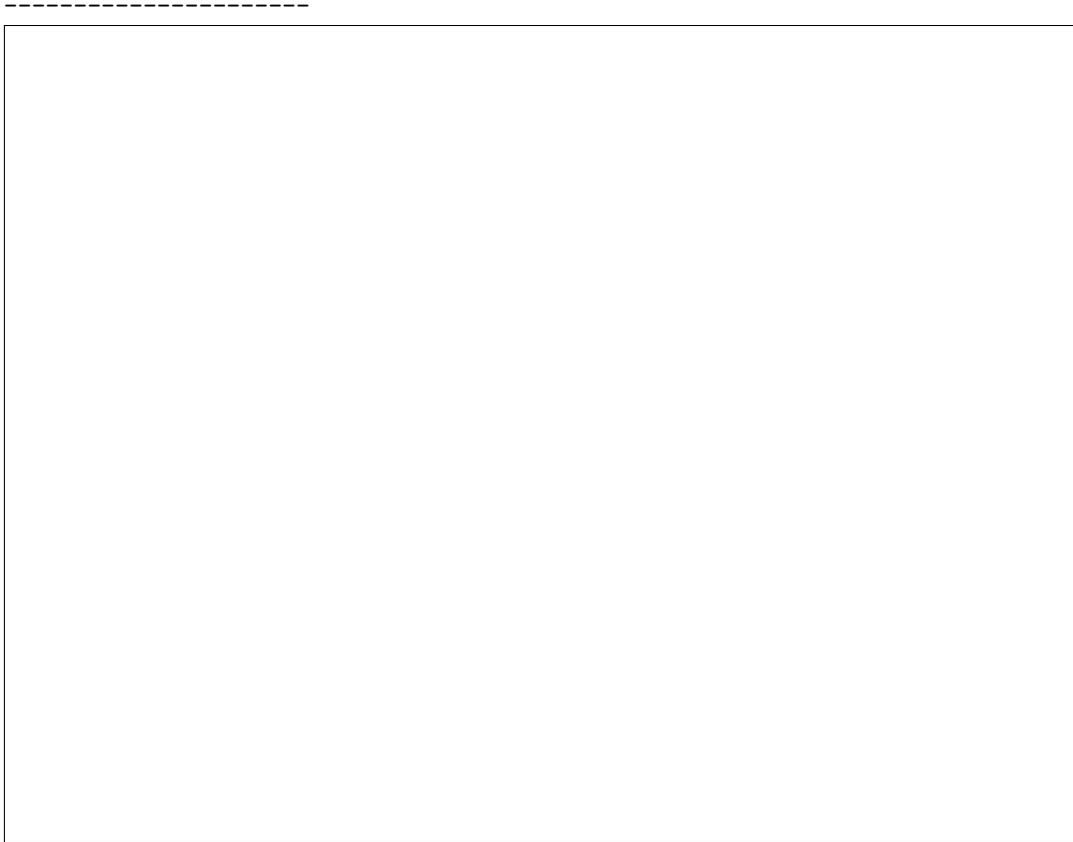
class Package:

    def __init__(self, item, weight):
        -----
        -----

class LargePackage(Package):

    -----

class SmallPackage(Package):
```



9. (10.0 points) Debugging

You are writing a function that takes in a list of pairs and a key. The first element of each pair is an integer which represents a key and the second element is a string which represents the value. The second argument to the function is a key. The function removes each of the pairs that has a key that matches the inputted key. Finally, it returns the values that were removed from the list, order of the returned values does not matter.

```
def test_func(pairs, key):
    """
    >>> lst = [[1, "A"], [2, "B"], [1,"C"]]
    >>> test_func(lst, 1)
    ['A', 'C']      #['C', 'A'] is also acceptable
    >>> lst
    [[2, 'B']]      #Mutate the original list

    >>> lst2 = [[8, "pop?"], [8, "pop!"], [8, "pop."]]
    >>> test_func(lst2, 8)
    ['pop.', 'pop!', 'pop?']  # any order of the three pops is fine
    >>> lst2
    []  #All of the pairs were removed
    """
```

1. output = []
2. for i in range(len(pairs)):
3. if pairs[i] == key:
4. output.append(pairs.pop(i))
5. return output

(a) For the following doctest what should the function return (assuming it has not been fixed yet)

```
>>> lst = [[1, "one"], [2, "two"], [2, "Two!"], [1, "One!"]]
>>> test_func(lst, 1)
```

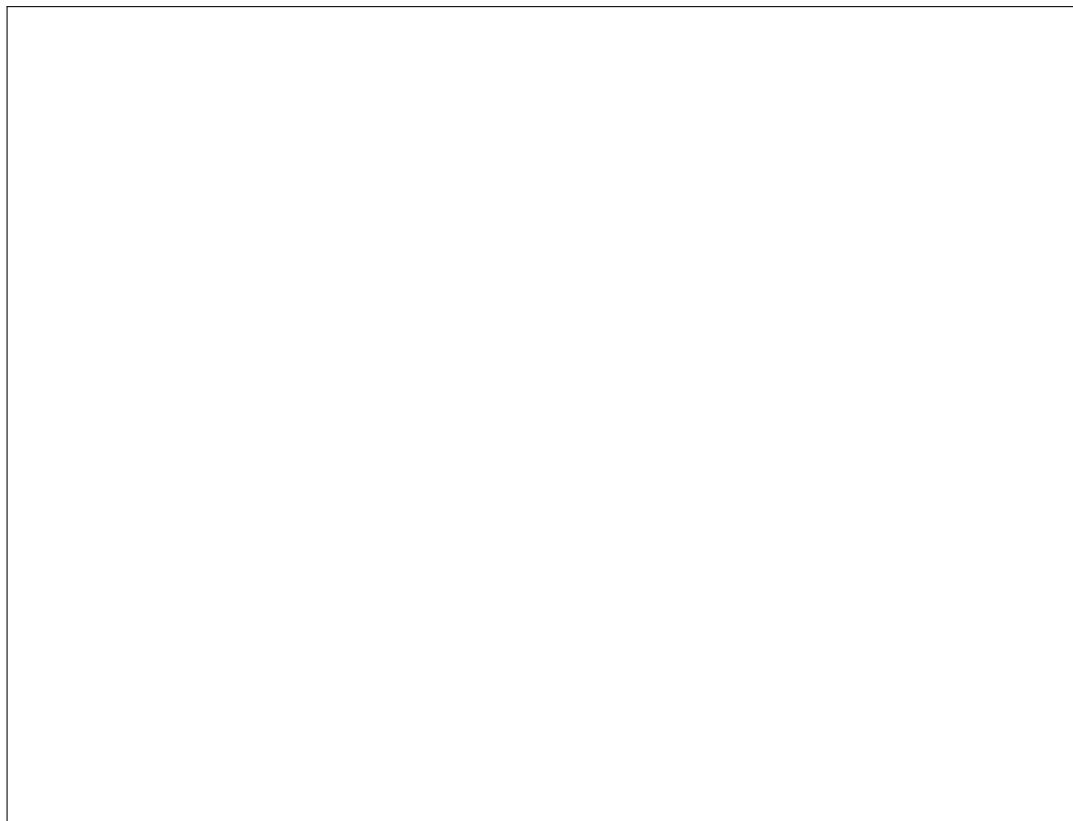
(b) (1.0 pt) What are the contents of `lst` after the doctest above has been run?

(c) (2.0 pt) For the input specified in part 1, what should (assume the function is now working as intended) the function return?

(d) (6.0 points)

Identify 3 bugs in the function. For each bug, include the line number, the code that is incorrect and describe the fix (a few words is enough). There are 3 distinction types of bugs in the question, but some of the bugs have multiple ways of fixing them. After fixing the bugs, the function should work correctly.

- i. (2.0 pt) Describe the first UNIQUE bug.



ii. (2.0 pt) Describe the secon UNIQUE bug.

A large, empty rectangular box with a thin black border, intended for the student to write their answer to the question. The box is currently blank.

iii. (2.0 pt) Describe the third UNIQUE bug

A large, empty rectangular box with a thin black border, intended for the student to write their answer to the question. The box is positioned below the question text and occupies a significant portion of the page's width and height.

(e) (3.0 pt) Now after identifying the bugs, write a fully working `test_func` function

A large empty rectangular box with a thin black border, intended for the student to write their code for the `test_func` function.

10. (10.0 points) One Giant Leap Forward, One Small Step Back

- (a) Jeff likes to traverse his LinkedLists weirdly. He likes to skip around a bit, specifically, jumping forward two and then jumping back one. As an example, if we had a linked list of:

1 -> 2 -> 3 -> 4 -> 5

Jeff wants 1 then 3 then 2 then 4 then 3 then 5, ending when we get to the last value in the linked list. Jeff would like you to create an iterator for him to iterate through any linked list this way, beginning with the first value in the linked list and ending with the last value.

The Link class is given to you as reference.

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

class LinkedListIterator:
    """
    >>> ll = Link(1, Link(2, Link(3, Link(4, Link(5))))
    >>> print(list(LinkedListIterator(ll)))
    [1, 3, 2, 4, 3, 5]
    """
    def __init__(self, lnk):
        self.pointer1 = lnk
        self.pointer2 = -----
        self.counter = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.pointer2 is Link.empty:
            -----

        if self.counter % 2 == 0:
            return_value = -----
            -----
        else:
            return_value = -----
            -----

        -----

        return return_value
```



No more questions.